

# Universidad Peruana de Ciencias Aplicadas - UPC

“Año de la universalización de la salud”



Carrera de Ciencias de la Computación - Curso de Inteligencia Artificial

Informe de la Aplicación

Trabajo 01 - Bin Packing de una Dimensión

Profesor: Shiguihara, Pedro

Presentado por: Villarruel Vásquez, Sergio Enrique

Monterrico, abril 2020

# CONTENIDO

<b>Planteamiento del problema a resolver</b>	<b>3</b>
<b>Hill Climbing</b>	<b>3</b>
<b>Heurística</b>	<b>3</b>
<b>Código fuente y recursos</b>	<b>3</b>
<b>Pruebas de uso</b>	<b>3</b>
<b>Bibliografía</b>	<b>3</b>

## 1. Planteamiento del problema a resolver

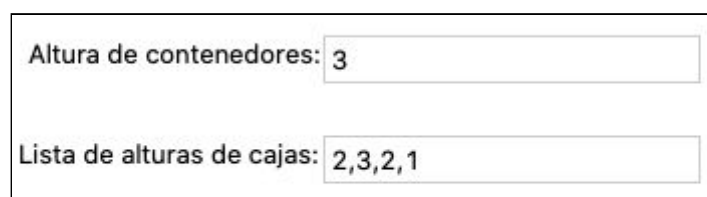
Partimos de un conjunto de contenedores y un conjunto de cajas que se deberán de colocar dentro de los contenedores. Los contenedores son todos iguales y se pueden llenar hasta cierta altura. Las cajas tienen la misma base que el contenedor, pero diferentes alturas, siempre por debajo de la altura de los contenedores. El problema a resolver se resume en: dado un conjunto de paquetes saber cual es el mínimo conjunto de contenedores necesario para poder almacenar las cajas.

Este problema es famosamente conocido como “El problema de la Mochila” y está diseñado de tal manera que encontrar una solución óptima para todos los posibles métodos de ordenamiento sobre un gran número de variables supondría un tiempo de ejecución demasiado amplio. Por lo tanto, la propuesta de resolver este problema con algoritmos de Inteligencia Artificial, nos daría la ventaja de reducir ampliamente este tiempo de ejecución obteniendo una muy buena solución, aunque no la mejor en algunos casos.

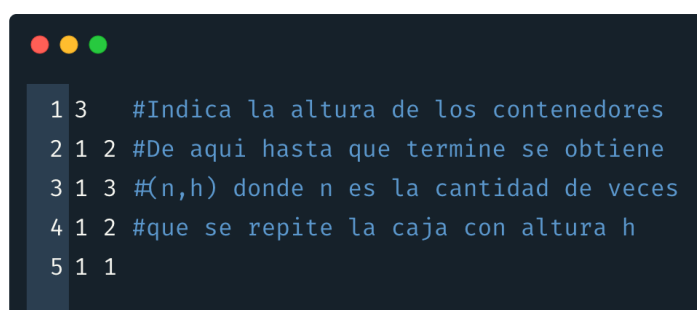
## 2. Propuesta de solución

Para poder otorgar una solución a este problema se ha diseñado una aplicación que implementa un algoritmo de búsqueda local usando las bases de la técnica de optimización **Hill-Climbing**.

Esta aplicación opera mediante una interfaz sencilla la cual recibirá entrada de datos de manera manual o mediante un formato de documento de texto específico.

Un formulario web con dos campos de entrada. El primer campo está etiquetado como 'Altura de contenedores:' y contiene el valor '3'. El segundo campo está etiquetado como 'Lista de alturas de cajas:' y contiene el valor '2,3,2,1'.

*Imagen 1 - Entrada de datos manual*

Una captura de pantalla de un editor de código con un fondo oscuro. Muestra un archivo de texto con cinco líneas de código, cada una precedida por un número de línea (1 a 5). Las líneas de código son: 1 3 #Indica la altura de los contenedores, 2 1 2 #De aquí hasta que termine se obtiene, 3 1 3 #(n,h) donde n es la cantidad de veces, 4 1 2 #que se repite la caja con altura h, 5 1 1.

*Imagen 2 - Formato de documento de texto*

Para poder ejecutar el algoritmo de optimización se tiene que elegir primero el estado inicial en el que empezará a recorrer la lista de cajas, estos estados iniciales pueden ser: mantener el orden de las cajas según la lista otorgada, ordenarlos ascendentemente o descendentemente.

Imagen 3 - Entrada de datos y configuración inicial

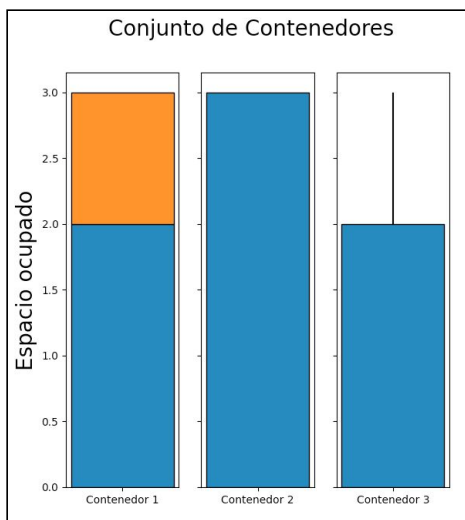
Una vez que se presione el botón de Optimizar, empieza la ejecución del algoritmo. Este algoritmo recibe cuál es el tamaño de los contenedores, así como la lista de alturas del conjunto de cajas. Y en resumen, lo que hace internamente es agrupar las cajas en un contenedor siguiendo una heurística específica, de tal forma que se pueda llegar a ocupar el espacio total de un contenedor. Y en caso de no poder insertar ninguna caja en el contenedor actual, creará un nuevo contenedor para agrupar las cajas restantes.

Por ejemplo si se tiene la siguiente información:

*Altura de los contenedores* = 3

*Lista de alturas de las cajas* = { 2, 3 , 2 , 1 }

El algoritmo insertará la primera caja de la lista en el primer contenedor, ya que se asume que la altura siempre será menor a la del contenedor, después de esto, va buscando a través de la lista, según la heurística, cuál es la mejor caja a insertar en el contenedor. En caso de exceder el límite crea un nuevo contenedor y el proceso se repite hasta que no existen cajas disponibles por insertar.



El resultado esperado sería:

*Contenedor 1* = {2,1}

*Contenedor 2* = {3}

*Contenedor 3* = {2}

### 3. Heurística

La función heurística que analiza este algoritmo se compone de la siguiente manera:

Si  $Altura_{Caja} + AlturaOcupada_{Contenedor} \leq Altura_{contenedor}$  entonces:  
 $h(Altura_{Caja}, AlturaOcupada_{Contenedor}) = Altura_{Caja} + AlturaOcupada_{Contenedor}$

Si  $Altura_{Caja} + AlturaOcupada_{Contenedor} > Altura_{contenedor}$  entonces:  
 $h(Altura_{Caja}, AlturaOcupada_{Contenedor}) = 0$

Ejemplo:

Altura Contenedor = 3

Lista de altura de Cajas = {3, 2, 1}

Altura Ocupada del Contenedor = 2

$h(3,2) = 0$  ya que  $2 + 3 > 3$

$h(2,2) = 0$  ya que  $2 + 2 > 4$

$h(1,2) = 3$  ya que  $2 + 1 \leq 3$

Posteriormente, después de haber calculado la función heurística para cada caja, seleccionaremos la que nos dé el mayor valor. Si hay más de un resultado que nos den el mismo dato seleccionaremos el que se encontró primero. La caja seleccionada es retirada de la lista de cajas y agregada al contenedor.

Si el resultado del máximo valor es 0, significa que ninguna de las cajas puede caber en el contenedor, por lo cual, según la información obtenida procedemos a crear un nuevo contenedor, y colocar la caja de la primera posición de la lista.

Esta función heurística nos podrá garantizar que se seleccionarán el conjunto de cajas adecuado para llenar al máximo cada uno de nuestros contenedores. Evitando así la creación de contenedores innecesarios y otorgandonos la optimización deseada.

## 4. Código fuente y recursos

La aplicación que he desarrollado se encuentra codificado con el lenguaje de programación Python. Ejecutado con la versión 3.7.4 del mismo.

Las librerías que he usado para su desarrollo son las siguientes:

- numpy - librería de funciones matemáticas de alto nivel
- tkinter - librería para generar una interfaz gráfica de usuario
- matplotlib - librería para generación de gráficos

Los diferentes archivos de scripts que componen la aplicación son:

- main\_app.py - Contiene la creación y la ejecución de la interfaz gráfica
- data\_in.py - Contempla lo necesario para la obtención de datos de manera correcta
- graphics.py - Contiene las funciones que muestran las gráficas de la colocación de cajas en los contenedores.
- hillclimbing\_binpacking.py - Contempla el algoritmo de optimización y los cálculos de la función heurística

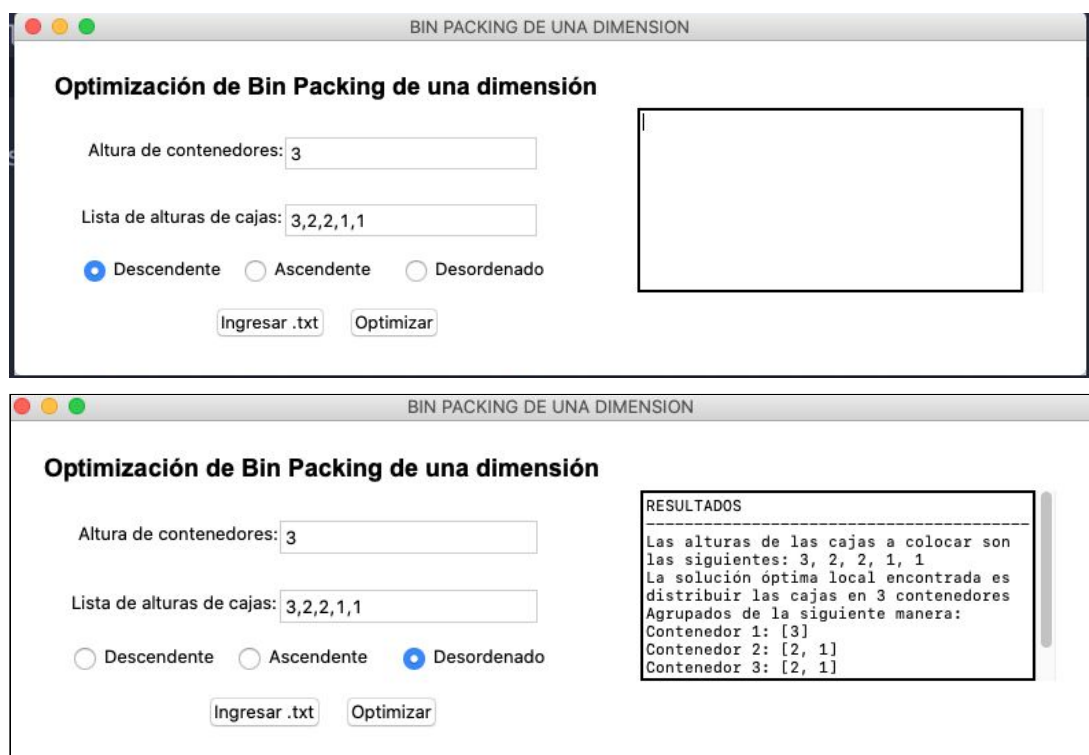
La aplicación ha sido desarrollada y testeada usando el editor de código Visual Studio Code. Adicionalmente, se ha verificado su compatibilidad tanto como para Windows 10 y macOS

## 5. Uso de la aplicación

En este apartado mostraré algunas de las funcionalidades de la aplicación implementadas y también pruebas con diferentes datasets.

### Ventana principal:

En la ventana principal se pueden insertar manualmente o mediante un archivo los datos de entrada, se puede seleccionar si se desea ordenar la lista de alturas, y al lado derecho se imprimirán de los resultados encontrados.



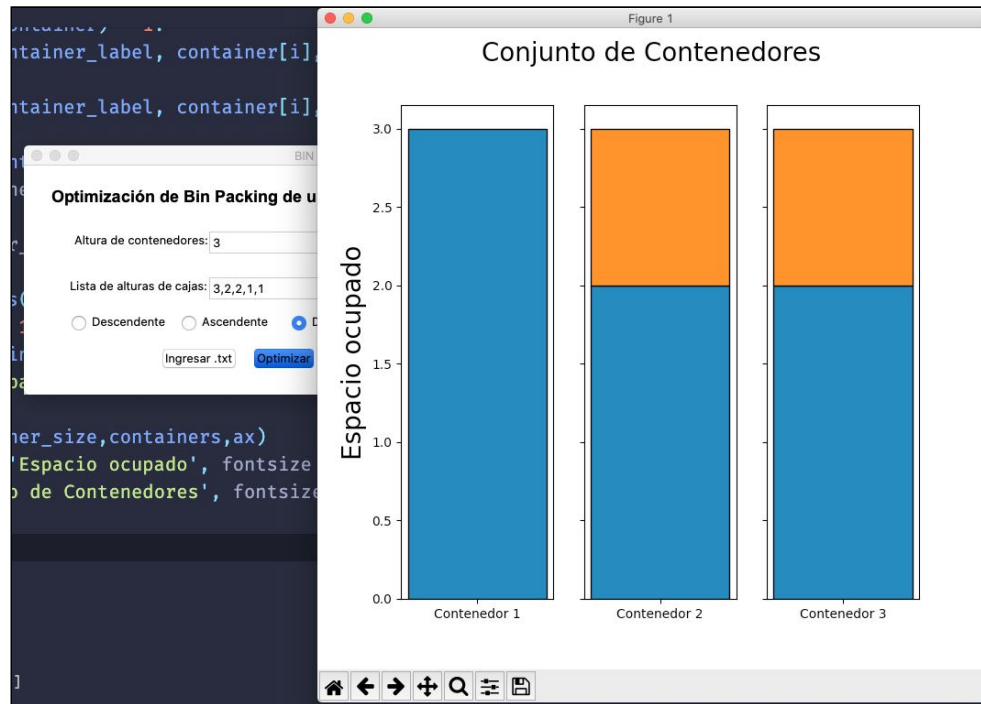
*Imagen 4 - Ventana Principal*  
**Código fuente: main\_app.py**

```
1 window=Tk()  
2 mywin=MyWindow(window)  
3 window.title('BIN PACKING DE UNA DIMENSION')  
4 window.geometry("800x250+10+10")  
5 window.mainloop()
```

*Código 1 - Creación de Ventana Principal*

### **Gráfico de los resultados:**

Después de accionar el botón de Optimizar, se mostrará un gráfico con los resultados obtenidos para poder visualizar la información.



*Imagen 5 - Gráfico de los resultados*

### **Código fuente: graphics.py**

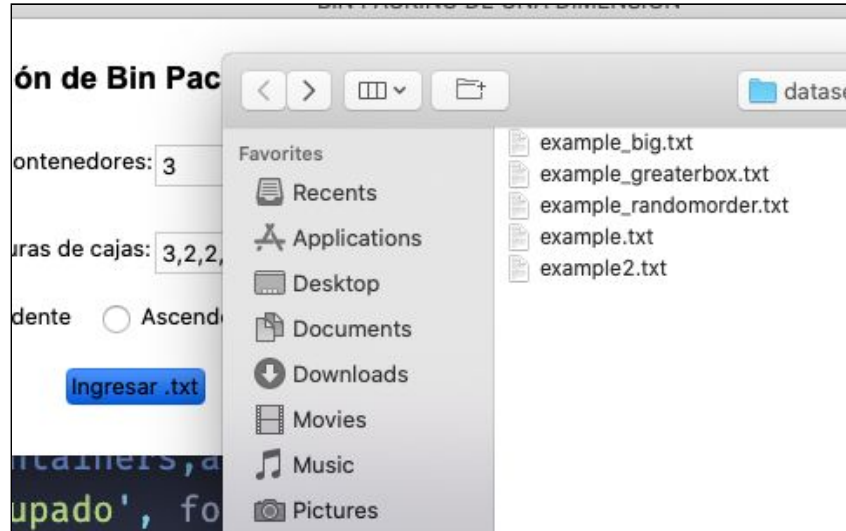
```
1 def show_results(container_size,containers):
2     fig, ax = plt.subplots(1,len(containers),figsize (10+len(containers),7),sharey=True)
3     if len(containers) == 1:
4         single_draw(container_size,containers,ax)
5         ax.set_ylabel('Espacio ocupado', fontsize = 20)
6     else:
7         multi_draw(container_size,containers,ax)
8         ax[0].set_ylabel('Espacio ocupado', fontsize = 20)
9     fig.suptitle('Conjunto de Contenedores', fontsize = 20)
10    plt.show()
```

*Código 2 - Gráfico de los resultados*



### Lectura de datos:

La aplicación puede leer documentos .txt con un formato específico para agilizar la entrada de datos, posteriormente se extraen los datos de manera que puedan usarse en nuestra aplicación.



### Código fuente: data\_in.py

```
1 def read(path):
2     """
3     Función para leer un archivo y obtener los valores correspondientes
4     retorna la capacidad del contenedor y el número de cajas
5     ordenadas de mayor a menor
6     """
7     #OBTENCIÓN DE ARCHIVOS
8     file = open(path,"r")
9     f1=file.readlines()
10    #FILTADO DEL ARCHIVO
11    data = []
12    boxes = []
13    for line in f1:
14        data.append([str(n) for n in line.split(' ')])
15    #SE OBTIENEN LOS DATOS
16    container_size = int(data[0][0])
17    for i in range(1,len(data)):
18        n_box = int(data[i][0])
19        for time in range(n_box):
20            valor = int(data[i][1])
21            if valor > container_size: return 0,0
22            boxes.append(valor)
23    return container_size, boxes
```

*Código 3 - Obtención y filtrado de datos de un .txt*

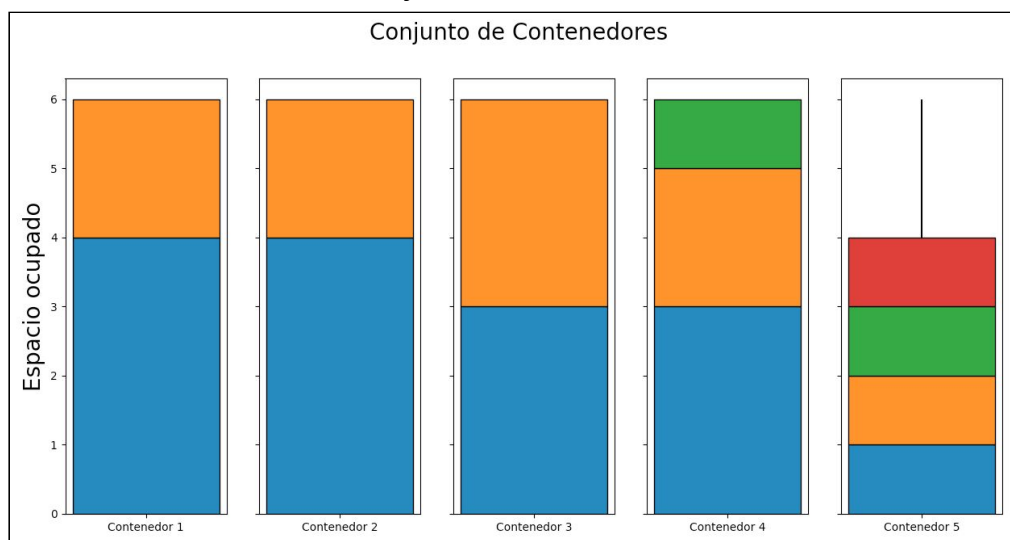
## Testing de la aplicación con diferentes datasets:

Para esta sección usaré 3 diferentes datasets:

### ***Dataset 1:***

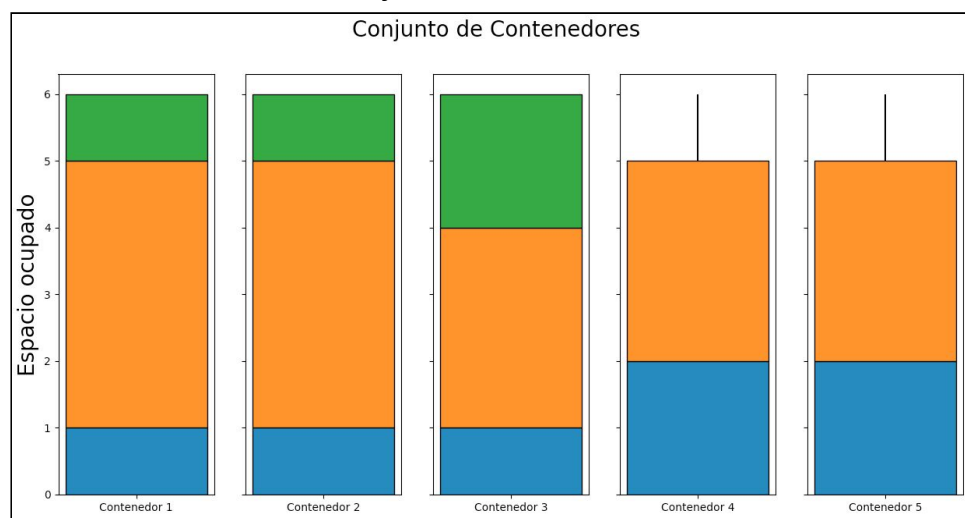
1	6	
2	2	4
3	3	3
4	3	2
5	5	1

### Orden inicial de cajas ordenadas descendentemente



### ***Resultado 1***

### Orden inicial de cajas ordenadas descendentemente

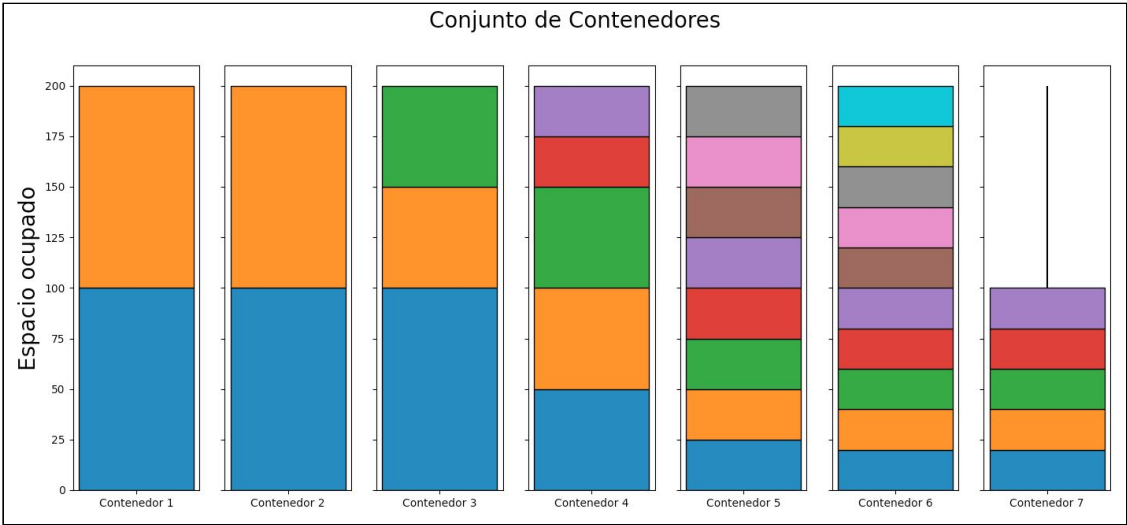


### ***Resultado 2***

**Dataset 2:**

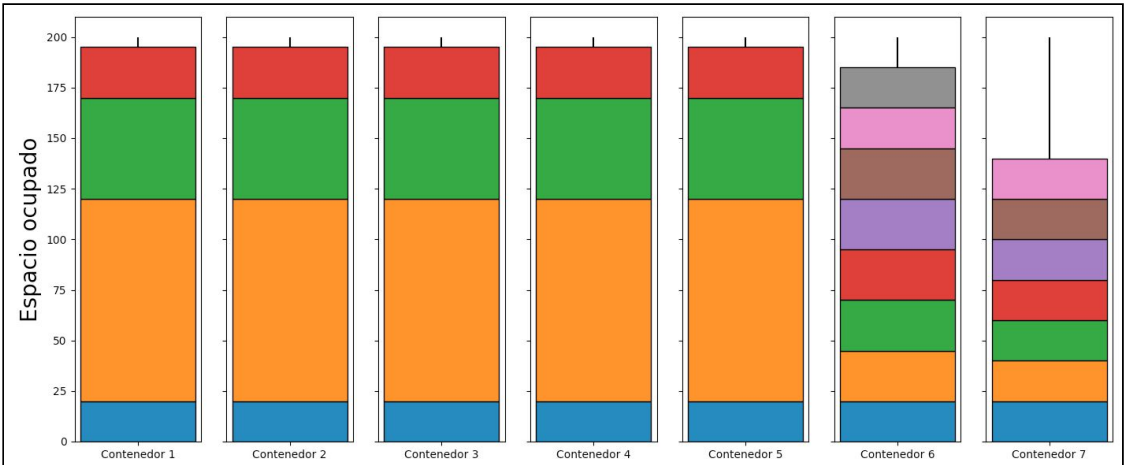
1	200
2	5 100
3	5 50
4	10 25
5	15 20

Orden inicial de cajas ordenadas descendientemente



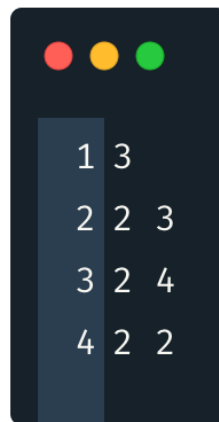
Resultado 3

Orden inicial de cajas ordenadas descendientemente



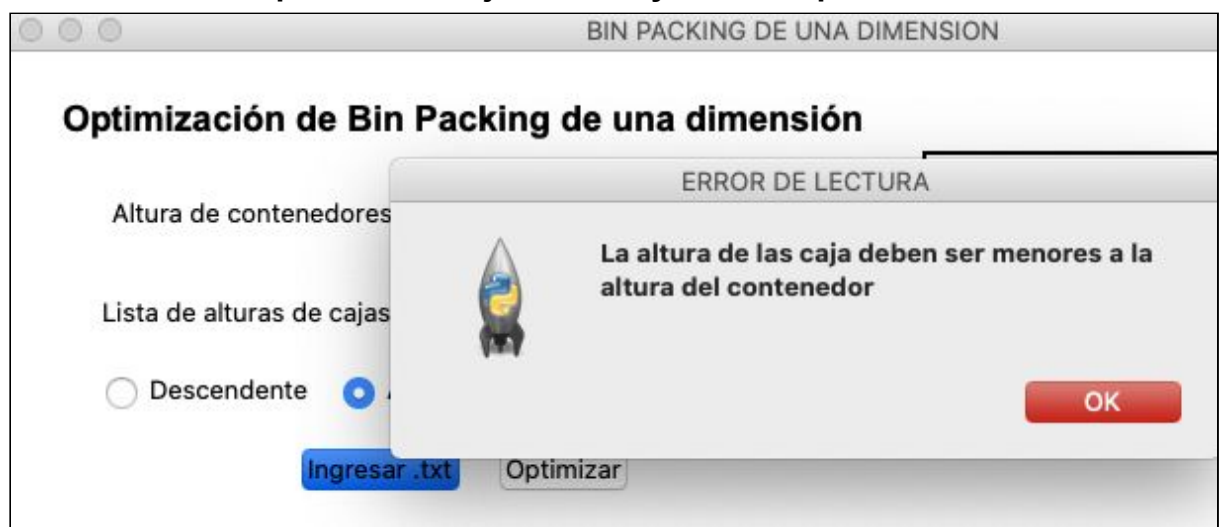
Resultado 4

**Dataset 3:**



1	3
2	2 3
3	2 4
4	2 2

**Error debido a que existen cajas con mayor altura que el contenedor**



*Resultado 5 - ERROR*

## 6. Bibliografía

2020. *Col·lecció De Problemes - Intel·ligència Artificial*. 1st ed. Catalunya: Universitat Politècnica de Catalunya, pp.5-33.

2012.F.D. Alves Bin packing and related problems: pattern-based approaches (tesis de maestría) Faculdade de Ciencias da Universidade do Porto, Portugal (2012)

2014.Pérez J., Castillo H., Vilariño D., De-la-Rosa R., Mexicano A., Zavala J.C., Santaolaya R., Estrada H. Metaheuristic for selecting lower bound applied to the problema of bin packing, Conference Proceedings International Conference on Innovative Trends in Science, Engineering and Management 2014 (ICITSEM2014), Dubai, UAE, pp. 196-201, 2014